SAT-based Decision Tree Learning for Large Data Sets

André Schidler and Stefan Szeider
Algorithms and Complexity Group, TU Wien, Vienna, Austria
aschidler@ac.tuwien.ac.at, sz@ac.tuwien.ac.at

Abstract

Decision trees of low depth are beneficial for understanding and interpreting the data they represent. Unfortunately, finding a decision tree of lowest depth that correctly represents given data is NP-hard. Hence known algorithms either (i) utilize heuristics that do not optimize the depth or (ii) are exact but scale only to small or medium-sized instances. We propose a new hybrid approach to decision tree learning, combining heuristic and exact methods in a novel way. More specifically, we employ SAT encodings repeatedly to local parts of a decision tree provided by a standard heuristic, leading to a global depth improvement. This allows us to scale the power of exact SAT-based methods to almost arbitrarily large data sets. We evaluate our new approach experimentally on a range of real-world instances that contain up to several thousand samples. In almost all cases, our method successfully decreases the depth of the initial decision tree; often, the decrease is significant.

Introduction

Decision trees are among the oldest, and most widely used tools for the description, classification, and generalization of data (Murthy 1998). Since decision trees are easy to understand and interpret, they can contribute to the general objective of explainable and interpretable AI (Gilpin et al. 2018). In this context, one prefers decision trees of low depth as they are easier to understand and interpret (Molnar 2019). In view of the parsimony principle, low depth decision trees have a better chance to generalize over additional samples (Bessiere, Hebrard and O'Sullivan 2009). Since low depth decision trees require fewer tests per sample, they are strongly preferred in applications like in medical diagnosis where tests might be costly, risky, or intrusive (Podgorelec et al. 2002).

Unfortunately, finding a decision tree of lowest depth that correctly classifies a given data set is NP-hard (Hyafil and Rivest 1976). Consequently, the standard heuristic methods for decision tree learning like C4.5 (Quinlan 1993) or ITI (Utgoff, Berkman and Clouse 1997), which are fast and scale to large data sets, do not find decision trees of lowest depth. Therefore, several exact methods have been proposed that encode the problem into SAT or CSP and use SAT/CSP solvers to find an optimal tree (Bessiere, Hebrard and O'Sullivan 2009; Narodytska et al. 2018; Avellaneda 2020; Janota and Morgado 2020). Indeed, in many cases, these exact methods produce decision trees that are significantly smaller in size or depth than the decision trees found by standard heuristic algorithms. However, the exact methods can only be applied to relatively small data sets. The currently best method is due to Avellaneda (2020). It is capable of producing decision trees of a depth up to 12.

Constraint programming has been used to optimize different properties of decision trees. While we search for trees of low depth that correctly classify the data set, these approaches try to minimize the number of misclassifications subject to constraints on the depth (Bertsimas and Dunn 2017; Verwer and Zhang 2019; Verhaeghe et al. 2020) or with penalties on trees with a large number of leaves (Hu, Rudin and Seltzer 2019).

In this paper, we propose a novel approach to learning decision trees of low depth. We combine the scalability of heuristic methods with the strength of encoding-based exact methods, thus taking the best of the two worlds. Our approach follows the principle of *SAT-based Local Improvement (SLIM)* which starts with a solution provided by a fast heuristic. It then repeatedly applies a SAT-based exact method locally to improve the solution. SLIM has shown to be effective in graph decomposition problems (Lodha, Ordyniak and Szeider 2017b; Fichte, Lodha and Szeider 2017; Lodha, Ordyniak and Szeider 2017a; Ganian et al. 2019) and Bayesian network structure learning (Ramaswamy and Szeider 2021). SLIM is similar to Large Neighborhood Search (Pisinger and Ropke 2010), where SLIM distinguishes itself by combining a structurally constrained notion of neighbourhood with a complete method (SAT).

Key to our approach is a suitable notion of a local instance I', generated from a subtree T' of a given decision tree T, so that we can utilize a new decision tree T'' for I' of lower depth than T' to reduce the depth of T. To make this work, we introduce new classification categories which guarantee that certain samples end up in the same leaves of T''. This allows us to insert subtrees copied from T at these leaves, obtaining a new global decision tree T^* (Theorem 1). By adding certain weights to the new classification categories, we can ensure that by decreasing the depth locally, we can eventually decrease the depth globally (Theorem 2).

Because of the new classification categories and the weights, a local instance poses a more complex classification

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

problem. The available SAT/CSP encodings only support binary unweighted classification instances and are therefore not directly applicable. We show, however, how SAT encodings can be generalized to accommodate non-binary weighted classification instances and propose a subtree selection strategy that avoids weights (Corollary 1). We further propose a new encoding based on a characterization of decision trees in terms of partitions (Theorem 3), which allows us to handle local instances of higher depth than it is possible with known encodings.

We establish a prototype implementation of our approach (DT-SLIM) and empirically evaluate it on data sets from the UCI Machine Learning Repository. Our experimental results are very encouraging: we can improve the depth of heuristically obtained decision trees in almost all cases, in some cases significantly. For instance, the decision tree for benchmark set "australian" computed by the standard heuristic Weka has a depth of 53, which DT-SLIM reduces to a depth of 22.

We also compare the test accuracy of the decision trees, before and after local improvement. The principle of Occam's Razor suggests that a decision tree of lower depth generalizes better to additional data. Our results affirm this suggestion. In a vast majority of the cases, deep decision trees generalize worse than their depth-improved counterparts. In several cases, DT-SLIM significantly increases the test accuracy; for instance, reducing the decision tree depth for benchmark set "objectivity" from 36 to 10 increases the test accuracy from 55% to 78%.

Preliminaries

Classification problems. An *example* (or *sample* or *feature vector*) e is a function $e : \mathit{feat}(e) \to \{0, 1\}$ defined on a finite set $\mathit{feat}(e)$ of *features* (or *attributes*). For a set E of samples, we put $\mathit{feat}(E) = \bigcup_{e \in E} \mathit{feat}(e)$. We say that two samples e_1, e_2 *agree* on a feature f if $f \in \mathit{feat}(e_1) \cap \mathit{feat}(e_2)$ and $e_1(f) = e_2(f)$. If $f \in \mathit{feat}(e_1) \cap \mathit{feat}(e_2)$ but $e_1(f) \neq e_2(f)$, then we say that the samples *disagree on* f.

A *classification instance* I is a pair (E, C) where E is a set of samples, for all $e_1, e_2 \in E$ we have $\mathit{feat}(e_1) = \mathit{feat}(e_2)$, and C is a mapping that assigns each sample $e \in E$ an integer $C(e)$, the *classification* of e. For a set $E' \subseteq E$ we let $C(E') = \{\, C(e) : e \in E' \,\}$.

An important special case are *binary* classification instances $I = (C, E)$ with $C(E) = \{0, 1\}$; here we call an $e \in E$ *negative* if $C(e) = 0$ and *positive* if $C(e) = 1$.

A set $E' \subseteq E$ of samples of a classification instance (C, E) is *uniform* if $|C(E')| \leq 1$, otherwise, E' is non-uniform.

A classification instance (C', E') is a *subinstance* of (C, E) if $E' \subseteq E$ and C' is the restriction of C to E'.

Given a classification instance (E, C), a subset $F \subseteq \mathit{feat}(E)$ is a *support set* of E if any two samples $e_1, e_2 \in E$ with $C(e_1) \neq C(e_2)$ disagree in at least one feature of F.

Decision trees. A *(binary) decision tree*, or *DT*, for short, is a rooted tree T with vertex set $V(T)$ and arc set $A(T)$, where each non-leaf node $v \in V(T)$ is labeled with a feature $\mathit{feat}(v)$ and has exactly two outgoing arcs, a *left arc* and a *right arc*. We write $\mathit{feat}(T) = \{\, \mathit{feat}(v) : v \in V(T) \,\}$. The

sample	sunny	rain	overcast	temp:mild	temp:hot	temp:cool	humid	windy	play golf
e_1	0	1	0	0	0	1	0	1	1
e_2	0	1	0	1	0	0	0	1	1
e_3	1	0	0	0	1	0	1	1	0
e_4	0	0	1	0	0	1	0	0	1
e_5	1	0	0	1	0	0	0	0	1
e_6	0	1	0	1	0	0	1	0	0
e_7	0	1	0	1	0	0	1	1	1
e_8	0	0	1	0	1	0	1	1	1
e_9	1	0	0	1	0	0	1	1	0
e_{10}	1	0	0	0	0	1	0	1	1
e_{11}	0	0	1	1	0	0	1	0	1
e_{12}	1	0	0	0	1	0	1	0	0

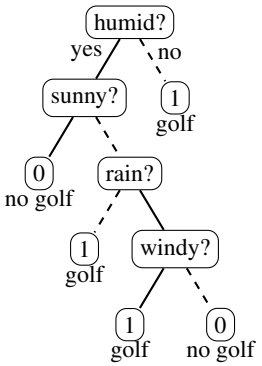

Figure 1: Left: A classification instance with 12 samples and 8 features, the last column indicating whether the sample is positive or negative. Right: a decision tree of depth 4 for the classification instance on the left.

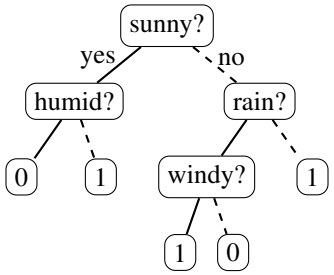

Figure 2: A decision tree of depth 3 for the classification instance in Fig. 1.

depth $d(T)$ of a decision tree T is the length of a longest path from the root to a leaf. $P_T(v)$ denotes the path from the root to the node v and $P_{T,i}(v)$ denotes the $i+1$-th node on this path, where $P_0(v)$ is the root. We also define the *depth of a node* $v \in V(T)$ *in* T, denoted $d_T(v)$, as the length of $P_T(v)$; clearly $d_T(v) \leq d(T)$.

Consider a classification instance (E, C) and a decision tree T with $\mathit{feat}(T) \subseteq \mathit{feat}(E)$. For each node v of T we define $E_T(v)$ as the set of all samples $e \in E$ such that for each left (right, respectively) arc (u, w) on $P_T(v)$ we have $e(f) = 1$ ($e(f) = 0$, respectively) for the feature $f = \mathit{feat}(u)$. We say that T *classifies* (E, C) (or simply that T is a decision tree *for* E) if $E_T(v)$ is uniform for each leaf v of T. If T classifies (E, C), then, slightly abusing notation, we write $C(v) = c$ if v is a leaf of T with $C(E_T(v)) = \{c\}$.

For a decision tree T and a node $v \in V(T)$, we denote by T_v the decision tree formed by the subtree of T rooted at v. If T classifies a classification instance (E, C), then T_v classifies the subinstance $(E_T(v), C')$.

Figure 1 shows an example for a classification problem and a corresponding decision tree. Figure 2 shows a decision tree of smallest depth for the same instance.

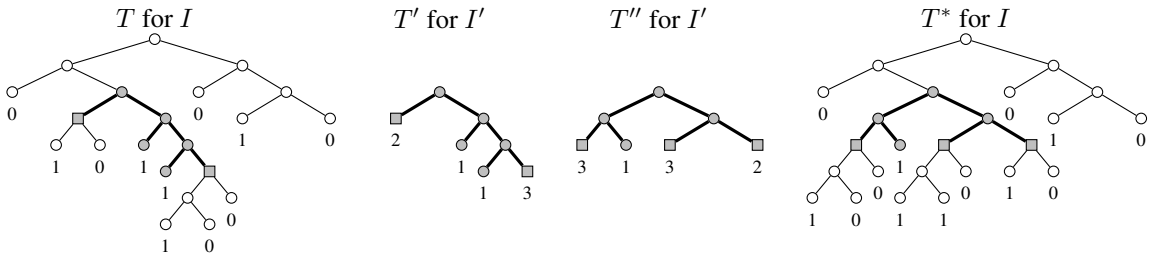

Figure 3: Local improvement workflow. The numbers indicate the leaves' classes; squares indicate special leaves.

Local Improvement

Assume we are given a classification instance $I = (E, C)$, which is too large to compute a decision tree of smallest depth for it using an exact method such as a SAT encoding. We can use a heuristic method to compute a non-optimal decision tree T for I. The idea of local improvement is to repeatedly select subtrees T' of T that induce a *local instance* I' that is small enough (possibly after further simplification and reduction) to be solved by an exact method. Once we have found a decision tree T'' for I' of smallest depth (or at least a depth that is smaller than the depth of T'), we can *replace* T' in T with the new T'', obtaining a new decision tree T^* for I.

However, to instantiate this general idea, we need to develop a suitable concept of a local instance. It must guarantee that when we have found a new local tree T'', we can extend each leaf of T'' with parts of T so that the overall decision tree T^* correctly classifies all the samples in $E_{T''}(v)$. The key to our solution is based on the introduction of new classes.

Let r be the root of T', let $\ell_1, \ldots, \ell_k$ be those leaves of T' that are not leaves of T, and let $s = \max_{e \in E} C(e)$. The *local instance associated with* T' is the pair $I' = (E', C')$ where $E' = E_T(r)$ and C' is the mapping defined by

$$C'(e) = \begin{cases} C'(e) = s + i & \text{if } e \in E_T(\ell_i) \text{ for some } 1 \leq i \leq k; \\ C'(e) = C(e) & \text{otherwise.} \end{cases}$$

Let T'' be any decision tree for I'. Obviously, T'' will contain for each $i \in \{1, \ldots, k\}$ at least one leaf m such that $C'(E'_{T''}(m)) = \{s + i\}$. We call such a leaf m a *special leaf with classification* $s + i$.

To describe how the new decision tree T^* is built together, we need the following operation on decision trees: Let T_1, T_2 be decision trees, x a leaf of T_1 and y the root of T_2. The *extension of* T_1 *at* x *with* T_2 is the decision tree T_3 obtained from T_1 and T_2 by taking the vertex-disjoint union of the two trees and identifying x with y.

To construct T^*, we start with the decision tree T_0 obtained from T by deleting all the descendants of the root r of T'. From T_0 we obtain T_1 by extending it at r by T''. Finally, from T_1 we obtain T^* by extending each special leaf m with classification $s + i$ with a new copy $T^m_{\ell_i}$ of T_{ℓ_i}. Figure 3 shows an example of this process.

The next theorem states that this replacement process is sound.

Theorem 1. *The decision tree T^* classifies I.*

Proof. For showing the claim, let ℓ^* be any leaf of T^*. We will show that $|C(\ell^*)| \leq 1$. Let P be the unique path in T^* from the root of T^* to ℓ^*. We distinguish several cases.

Case 1: P does not run through r. Hence ℓ^* is also a leaf of T. Since T correctly classifies I by assumption, $1 \geq |C(E_T(\ell^*))| = |C(E_{T^*}(\ell^*))| = |C(\ell^*)|$.

Case 2: P runs through r.

Subcase 2.1: ℓ^* is a leaf of T''. Since ℓ^* is also a leaf of T^*, it isn't a special leaf. The latter implies that $C(\ell^*) = C'(\ell^*)$. Since T'' correctly classifies I', we have $|C'(\ell^*)| \leq 1$, hence again $|C(\ell^*)| \leq 1$.

Subcase 2.2: ℓ^* is not a leaf of T''. Consequently, P runs through a special leaf m of T''. Let $s + i$ be the classification of m. By construction, the subtree T^*_m of T^* is a copy of the subtree T_{ℓ_i} of T, and the leaf ℓ^* of T^*_m is the copy of a leaf ℓ of T^*_m. Since $C'(m) = \{s + i\}$, we have $E_{T^*}(m) \subseteq E_T(\ell_i)$. Consequently $E_{T^*}(\ell^*) \subseteq E_T(\ell)$. Since T correctly classifies I, $|C(\ell)| \leq 1$, and from $E_{T^*}(\ell^*) \subseteq E_T(\ell)$ we thus get $|C(\ell^*)| \leq 1$. □

Let us now turn to the question of decreasing the depth of the input decision tree T employing such a local replacement. This does not work out of the box: Even when $d(T'') < d(T')$ it still can happen that $d(T^*) > d(T)$, since the depth of a special leaf v of T'' of classification $s + i$ can be larger than the depth of the corresponding leaf ℓ_i of T', resulting in a larger depth of T^* if the subtree attached to v at T^* is large.

To overcome this problem, we enrich the local instance by additional information, defining a weighted version of the classification problem.

Weighted classification. A *weighted classification instance* is a tuple $I_w = (E, C, d)$ where $I = (E, C)$ is a classification instance, and d is a mapping that assigns each $c \in C(E)$ a non-negative positive integer $d(c)$. I and I_w have the same decision trees, just the depth for decision trees are defined differently for I and I_w. Consider a decision tree T for I_w. For a leaf ℓ of T with classification c (i.e., $C(\ell) = \{c\}$), we define the *weighted depth of* ℓ *in* T as $d_{w,T}(\ell) = d_T(\ell) + d(c)$. The *weighted depth* $d_w(T)$ of T is the maximum weighted depth over all its leaves.

We will show how locally decreasing the weighted depth of the weighted local instance within our local improvement setting allows us to decrease the depth of the global decision tree.

Let $I = (E, C)$, $I' = (E', C')$, T, T', T'' and T^* as above, and let $I'_w = (E', C', d)$ denote the *weighted local instance*, where the weights for $c \in C'(E')$ are defined as follows: if $c = s + i$ then $d(c) = d(T_{\ell_i})$; if $c \leq s$, then $d(c) = 0$. We note that T' is a decision tree of the weighted local instance and hence $d_w(T')$ is defined.

Theorem 2. *If $d_w(T'') \leq d_w(T')$ then $d(T^*) \leq d(T)$.*

Proof. Assume $d_w(T'') \leq d_w(T')$ and consider a longest path P^* in T^* between the root of T^* and a leaf ℓ^* of T^*.

If P^* does not pass through r, the root of T'', then it is also a root-to-leaf path of T, and so $d(T^*) = L(P^*) \leq d(T)$, and the claim of is established.

It remains to consider the case where P^* passes through r. Let P be a longest path in T which passes through r. Consequently, $L(P) \leq d(T)$.

We can write $L(P^*) = L_0^* + L_1^* + L_2^*$ where L_0^* is the length of the part of P^* between the root of T^* and r, L_1^* is the length of the part of P^* between r and a leaf of T'', and L_2^* is the length of the part of P^* between a leaf of T'' and ℓ. It is possible that $L_2^* = 0$.

Similarly, we can write $L(P) = L_0 + L_1 + L_2$, where the three integers are defined similarly, using L_1 for the length of the part of P inside T'.

By the definition of the weights, we have $L_1 + L_2 = d_w(T')$, and $L_1^* + L_2^* = d_w(T'')$. Since $d_w(T'') \leq d_w(T')$, $L_1^* + L_2^* \leq L_1 + L_2$. Since $L_0^* = L_0$ by construction, this gives $d(T^*) = L(P^*) \leq L(P) \leq d(T)$, as claimed. □

We now identify a special case of Theorem 2 where we only need to consider the unweighted local instance and still ensure that $d(T^*) \leq d(T)$. Let us call a subtree T' of T to be *safe* if for every leaf ℓ of T' it holds that $d(T') \leq d_w(T') - d(\ell)$.

Corollary 1. *If T' is safe and $d(T'') \leq d(T')$ then $d(T^*) \leq d(T)$.*

Proof. Let T' be a safe subtree with $d(T'') \leq d(T')$. Let ℓ'' be a leaf of T'' with $d_{w,T''}(\ell'') = d_w(T'')$ and let c be the classification of ℓ'' in T''. There must be a leaf ℓ of T' with classification $c = d(\ell)$. From the definitions we get $d_w(T'') = d_{w,T''}(\ell'') = d(c) + d_{T''}(\ell'') \leq d(c) + d(T'') \leq d(c) + d(T')$. Since T' is safe, we have $d(T') \leq d_w(T') - d(c)$, and so we get from $d_w(T'') \leq d(c) + d(T')$ that $d_w(T'') \leq d(c) + d_w(T') - d(c) = d_w(T')$. By Theorem 2, $d(T^*) \leq d(T)$ follows. □

SAT Encodings

The use of SAT encodings to induce decision trees has gained increased attention in recent times (Bessiere, Hebrard and O'Sullivan 2009; Narodytska et al. 2018; Avellaneda 2020; Janota and Morgado 2020). Out of all these encodings, *DT_depth* by Avellaneda (2020) performed best for our purposes. We give a brief outline of this encoding and refer to the original paper for further details. The idea behind DT_depth is to encode a complete binary tree of a specific depth. Constraints assigning each internal node exactly one feature and each leaf exactly one class are used. The decision tree is made consistent with the dataset as follows: for each sample and node it is encoded which feature the node must have assigned for the sample to use it. The same is done for leaves and classes. This idea allows the decision tree structure to be encoded efficiently. Given a depth d the encoding requires $\Omega(2^d)$ many clauses (Avellaneda 2020). The encoding can easily be extended to support more than two classes. This increases the number of clauses by a factor of $\log_2(|C(E)|)$. The encoding does not support weights without significant changes. Since the number of clauses is exponential in the depth, there is an upper bound on the depth that can be feasibly encoded using DT_depth. To allow the exploration of subtrees of larger depths, we propose a new encoding that significantly outperforms the existing encodings in that respect.

Our new encoding DT_pb. The idea behind our encoding is to formulate the problem in terms of partitions. This approach has been used successfully for different graph-related problems and was introduced by Heule and Szeider (2015) for clique-width computation. We first reformulate the problem of finding a decision tree with a given depth for a classification instance I by partitioning the set of samples (Theorem 3). We then directly convert this definition into a propositional CNF formula $\varphi(I, d)$, that is satisfiable if and only if a decision tree of depth d that classifies the instance I exists (Theorem 4).

Let $I = (E, C)$ be a classification instance and $\mathcal{S} = (S_0, \ldots, S_d)$ a sequence of partitions of E that has length d. We refer to the equivalence classes as *groups*. $\mathcal{S}$ is a *DT-sequence which classifies* $\mathcal{C}$ if the following conditions hold.

DT1 $S_0 = \{E\}$.

DT2 For all $1 \leq m \leq d$ it holds that, for each group $g \in S_{m-1} \setminus S_m$, there are groups $g', g'' \in S_m$ with $g = g' \cup g''$, such that for some $f \in \mathit{feat}(E)$, $e'(f) = 0$ for all $e' \in g'$ and $e''(f) = 1$ for all $e'' \in g''$.

DT3 For each $g \in S_d$ it holds that for all $e_1, e_2 \in g$: $C(e_1) = C(e_2)$.

We note that the definition implies that S_m is a refinement of S_{m-1}, for $1 \leq m \leq d$. The definition of DT-sequences corresponds to the definition of $E_T(v)$ and it is easy to see that decision trees can be converted into DT-sequences of the same depth; and the other way around. This leads us to the following theorem.

Theorem 3. *A classification instance can be classified by a decision tree of depth d if and only if it can be classified by a DT-sequence of length d.*

We encode a DT-sequence of length d for $I = (E, C)$ where $E = \{e_1, \ldots, e_n\}$ and $F = \mathit{feat}(E) = \{f_1, \ldots, f_k\}$. The result of our encoding is a propositional formula $\varphi(I, d)$. This formula is satisfiable if and only if there is a DT-sequence of length d, and therefore a decision tree of depth d exists, that classifies I.

We use the variables $g_{i,j,m}$, for $1 \leq i < j < n$, $0 \leq m \leq d$, with the semantics that $g_{i,j,m}$ is true if and only if samples e_i and e_j are in the same group at level m. We also use the variables $s_{i,m,\ell}$ for $1 \leq i \leq n$, $0 \leq m \leq d$, $1 \leq \ell \leq k$ to ensure DT2 is satisfied.

At the start, i.e., depth 0, all samples belong to the same group. We add the unary clause $g_{i,j,0}$ for all $1 \leq i < j \leq n$. At the last level, all samples in one group must belong to the same class. We enforce this by adding the unary clause $\neg g_{i,j,d}$ for all $1 \leq i < j \leq n$, such that e_i and e_j belong to different classes.

As S_m is a refinement of S_{m-1}, we have to ensure that samples in different groups can not be in the same group at a higher level. We state this by adding the clause $g_{i,j,m} \vee \neg g_{i,j,m+1}$ for all $1 \leq i < j \leq n$, $0 \leq m < d$.

In order to verify that DT2 holds, we must first ensure that at each level m, for every sample i there exists a corresponding feature to satisfy. For this purpose, we add the clause $\bigvee_{1 \leq \ell \leq k} s_{i,m,\ell}$ for $1 \leq i \leq n$, $0 \leq m < d$ and ensure consistency within groups by adding the clauses $\neg g_{i,j,m} \vee \neg s_{i,m,\ell} \vee s_{j,m,\ell}$ for $1 \leq i < j \leq n$, $0 \leq m < d$, $1 \leq \ell \leq k$.

We can now encode DT2 using the following clauses. For all $1 \leq i < j \leq n$, $0 \leq m < d$, $1 \leq \ell \leq k$: if $e_i(f_\ell) = e_j(f_\ell)$ we add the clause $\neg g_{i,j,m} \vee \neg s_{i,m,\ell} \vee g_{i,j,m+1}$, otherwise we add the clause $\neg s_{i,m,\ell} \vee \neg g_{i,j,m+1}$.

By construction of the formula and from Theorem 3 we obtain the following result.

Theorem 4. *$\varphi(I, d)$ is satisfiable if and only if there exists a decision tree of depth at most d that classifies I.*

The number of clauses in $\varphi(E, d)$ is $O(|E|^2 \cdot |\mathit{feat}(E)| \cdot d)$. While most of these clauses are binary or ternary, the number of literals per clause is in $O(|\mathit{feat}(E)|)$. Therefore, the main factor determining the encoding size is the number of samples and not the depth.

Our encoding excels for instances that require deep decision trees but have few samples. In comparison to DT_depth, DT_pb can solve instances that require deep and unbalanced decision trees. DT_depth encodes a complete tree of the required depth, which causes an exponentially large encoding, while DT_pb will remain comparatively small as long as the number of samples is small.

We can encode weights with DT_pb by using different maximum depths for the different classes. Let $d_{\min}$ be the lowest weight among all classes. Given a class c, for all $e_i, e_j \in E$ such that $C(e_i) = c, C(e_j) \neq c$, we add the clauses $\neg g_{i,j,w}$, where $w = d - d(c) + d_{\min}$ is the allowed depth in regards to the weight of c.

Subtree Selection and Feature Reduction

In this section, we describe the overall algorithm that facilitates the SAT-based local improvement, building upon the theoretical results of Section and the encodings described in Section .

As before, let T be a decision tree for a classification instance $I = (E, C)$. Our aim is to select a subtree T' which gives rise to a local instance $I' = (E', C')$ (and a weighted local instance $I'_w = (E', C', d)$). Since we will try to find a shorter decision tree T'' for I'_d with a SAT encoding, we need to select T' in such a way that the encoding size remains feasible.

Before we encode I'_w we further simplify it through *feature reduction*. We select a *support set* (recall the definition from Section) $F \subseteq \mathit{feat}(I'_w)$ and consider the new classification instance $R_F(I'_w) = (R_F(E'), R_F(C'), d)$ where $R_F(E') = \{ e|_F : e \in E' \}$ (the restrictions of the samples from E' to F) and $R_F(C')$, which is defined by $R_F(C')(e|_F) = C'(e)$. The latter definition is sound since F being a support set guarantees that for any two samples $e_1, e_2 \in E'$, if $C'(e_1) \neq C'(e_2)$, then $C'(e_1|_F) \neq C'(e_2|_F)$. Since $C'(E') = R_F(C')(R_F(E'))$, we can keep the same weighting d.

Observation 1. *If F is a support set for I'_w, and T'' is a decision tree for $R_F(I'_w)$, then T'' is also a decision tree for I'_w.*

We note, however, that not every decision tree for I'_w is necessarily a decision tree for $R_F(I'_d)$, and so with feature reduction we might lose depth-optimality.

With fewer features, $R_F(I'_w)$ can have fewer samples than I'_w since several samples (with the same classification) may collapse to a single sample if they agree on all the features in F.

Support sets can be determined in several ways. A natural choice is to take $F = \mathit{feat}(T')$, i.e., just keeping precisely those features used by the subtree T'. It is easy to see that F is indeed a support set for I'_w. This way, we can compute the support set quickly and often significantly reduce the number of samples. However, such a support set limits the improvement options to a rearrangement of the nodes in T'. As an alternative, we use other heuristic methods for determining support sets, with the potential of discovering completely different subtrees, but with a higher computational cost.

We use two heuristic methods: (i) Starting with $F = \mathit{feat}(I'_w)$, we iteratively remove one feature after the other. After each removal, we check whether F is still a support set. In case F is not a support set, we undo the removal and continue. (ii) As discussed by Boros et al. (2003), we start with $F = \emptyset$ and check for each pair $e_1, e_2 \in E'$ such that $C(e_1) \neq C(e_2)$ whether F contains a feature in which e_1 and e_2 disagree on; if not, we add such a feature to F . We compute support sets using both methods and use the smallest.

For *subtree selection*, we proceed as follows. We first select a node r of T as the root of T' and then "grow" the subtree step by step, adding one node after the other, as long as the depth of T' remains below some predefined number $\hat{d}$, and the number of samples in $R_F(I'_w)$ remains below some predefined number $\hat{c}$. The algorithm proceeds in a greedy fashion: in each iteration, it adds all nodes $u \in V(T_r) \backslash V(T')$ where $d(T_u)$ is maximal. This method has two advantages. First, it creates unbalanced trees that can then be balanced by the SAT solver to reduce the maximal depth. Second, it selects only safe subtrees, avoiding the requirement of weights, which simplifies the encoding.

We can now formulate the entire algorithm, which we refer to as DT-SLIM($\mathcal{H}$), where $\mathcal{H}$ denotes the heuristics used to generate the initial decision tree T. The pseudo-code for DT-SLIM($\mathcal{H}$) is shown in Algorithm 1. It iteratively selects a leaf v with maximum depth, ignoring those in the list of completed nodes D. The first subtree used to create a new instance is rooted at node $u \in P_T(v)$, such that u is the node

Algorithm 1: DT-SLIM.

Data: An instance $I = (C, E)$, a decision tree $T = (V, A)$, a depth limit $\hat{d}$, a sample limit $\hat{c}$.
Result: A new decision tree T and local instance I

$D \leftarrow \emptyset$
$T^* = T$
while $V(T^*) \setminus D \neq \emptyset$ **do**
  $v \leftarrow \arg\max_{v \in V(T) \setminus D} d(v)$
  $i \leftarrow \min\{\, i : |E_T(P_{T^*,i}(v))| \leq \hat{c} \vee i = d(v) \,\}$
  if $i < d(v)$ **then**
    $T', I' = \texttt{subtree}(P_{T^*,i}(v), I, T^*, \hat{d}, \hat{c}, 0, 0)$
    if $T'' = \texttt{compute\_dt}(I', d(T') - 1)$ **then**
      $T^* = \texttt{replace}(T^*, T', T'')$
  while $i > 0$ and $T = T^*$ **do**
    $i \leftarrow i - 1$
    for $l \in \{1, 0\}, r \in \{1, 2\}$ **do**
      $T', I' = \texttt{subtree}(P_{T^*,i}(v), I, T^*, \hat{d}, \hat{c}, r, l)$
      if $T'' = \texttt{compute\_dt}(I', d(T') - 1)$ **then**
        $T^* = \texttt{replace}(T^*, T', T'')$
        break
  if $T = T^*$ **then**
    $D \leftarrow D \cup P_{T^*}(v)$

closest to the root with $|E_T(u)| \leq \hat{c}$. It uses this instance without any further reductions ($r = 0$). The algorithm then proceeds up the path. For each node it tries to find a subtree using the following methods in the given order. First, DT-SLIM tries to find a subtree whose leaves are also leaves of T ($l = 1$) and then allows for more general subtrees. Each time, the algorithm first tries to reduce the instance using the features *feat*(T') ($r = 1$) and then uses a heuristically computed support set ($r = 2$).

Experiments[1]

Instances. We take all classification instances from the UCI Machine Learning Repository[2] that use discrete domains and contain more than 500 samples. We convert these instances to a binary domain by assigning a binary identifier to each distinct value. Additionally, we take the instances from Narodytska et al. (2018)[3], which were already used before (Bessiere, Hebrard and O'Sullivan 2009; Olson et al. 2017). In total, we use 37 different instances, which vary in the number of features (8 to 2195), the number of samples (11 to 23954), and the depth of the heuristic decision tree (3 to 97).

Accuracy. We compare the accuracy of the decision tree before and after local improvement on a set of samples for which the tree was not optimized. The *accuracy* is the percentage of these new samples correctly classified by the decision tree. Four of the considered instances already come with a designated test set. For all the other instances we use *5-fold stratified cross-validation* (also known as rotation estimation (Breiman et al. 1984; Kohavi 1995)) and report the average. We therefore run the experiments on 169 distinct instances.

The heuristic decision trees were generated by the tools ITI (Utgoff, Berkman and Clouse 1997) and Weka, the latter using the C4.5 algorithm (Quinlan 1993). The tools are configured to compute unpruned decision trees with 100% accuracy on the test set. Using unpruned trees as the input for DT-SLIM allows us to provide an accurate analysis of DT-SLIM's impact. One can still prune afterward to avoid overfitting for the prize of reduced accuracy on the training set. The same configurations have been used for ITI by Narodytska et al. (2018) and for Weka by Avellaneda (2020).

Setup. We use servers with two Intel Xeon E5-2640 v4 CPUs running at 2.40 GHz and using Ubuntu 18.04. The memory limit for each run is 8 GB and we use a timeout of 12 hours for the whole instance. We use the SAT solver Glucose 4.1[4] and the well-established decision tree inducers ITI 3.1[5] and Weka 3.8.4[6].

Results. We use a hybrid encoding which combines DT_depth and DT_pb. As previously discussed, up to a depth limit DT_depth performs better than DT_pb, but for larger depths, DT_pb performs better or is the only encoding that can handle such depths. To find the optimal threshold for switching encodings, we designed the following experiment. We artificially create simple instances that require a decision tree of a chosen depth. We increase this depth incrementally and observe how the runtime develops. The experiment shows that depth 10 is the best value for switching.

We establish the parameter values $\hat{d}$ and $\hat{c}$ as follows. Since the number of samples the encoding can handle depends on the depth, we do not use a single value $\hat{c}$, but overload $\hat{c}$ as a mapping from a depth value d to a sample limit. The goal is, given a time limit t, to find for each depth d a sample limit $\hat{c}(d)$ such that the SAT solver will find an improved subtree within t seconds for most instances. $\hat{d}$ is then implicitly defined as the maximum d such that $\hat{c}(d) > 0$. We refer to $\hat{d}$ and $\hat{c}$ for a given t as d_t and c_t respectively. In order to find these limits, we run DT-SLIM on selected instances and increase $\hat{d}$ and $\hat{c}$ incrementally from 3 to 49 and 10 to 1000 respectively, giving the SAT solver 900 seconds to find an improved subtree. We measure the time it takes to produce either an improved subtree (SAT), to determine that this is not possible (UNSAT), or reach the time limit. From the results we derive three sets of parameters: (i) $d_{60} = 12$ and c_{60} ranges from 70 to 15, (ii) $d_{300} = 15$ and c_{300} ranges from 300 to 90, and (iii) $d_{800} = 39$ and c_{800} ranges from 500 to 105.

The different parameter sets produce different results, as shown in Table 1 for decision trees generated by Weka and in Table 2 for decision trees generated by ITI. We present detailed results for selected instances in Tables 3 and 4.

We quantify the impact of our new encoding by running DT-SLIM on all instances using only DT_depth and the parameters d_{300}, c_{300}, as they match the limits of DT_depth the closest. We refer to this modified version as DT-SLIM*

[1] Source code can be found at https://github.com/ASchidler/decision_tree and results at https://doi.org/10.5281/zenodo.4571570.

[2] https://archive.ics.uci.edu/ml/datasets/

[3] We thank the authors for providing these instances.

[4] https://www.labri.fr/perso/lsimon/glucose/

[5] https://www-lrn.cs.umass.edu/iti/

[6] https://www.cs.waikato.ac.nz/~ml/weka/

	Parameters	Depth Av.	Depth Var.	Test Acc. Av.	Test Acc. Var.	Imp	Best
	Weka	22.63	436.76	0.67	0.10	–	–
SLIM	d_{60}, c_{60}	13.64	244.42	0.75	0.06	36	1
SLIM	d_{300}, c_{300}	13.45	261.29	0.76	0.05	36	6
SLIM	d_{800}, c_{800}	14.33	318.01	0.76	0.05	35	15

Table 1: DT-SLIM(Weka): Comparison of the changes regarding depth and test accuracy based on the choice of timeout and corresponding limits. *Imp* shows how many of the 37 decision trees could be improved, and *Best* shows for how many instances DT-SLIM found the lowest depth using this setting. The first row shows the baseline.

	Parameters	Depth Av.	Depth Var.	Test Acc. Av.	Test Acc. Var.	Imp	Best
	ITI	10.75	28.80	0.77	0.45	–	–
SLIM	d_{60}, c_{60}	8.55	22.99	0.77	0.04	36	1
SLIM	d_{300}, c_{300}	8.35	22.35	0.78	0.03	36	6
SLIM	d_{800}, c_{800}	8.59	29.60	0.77	0.03	35	15

Table 2: DT-SLIM(ITI): As in Table 1.

and compare it to DT-SLIM using the same parameters. DT-SLIM(Weka) finds decision trees with a depth at least as low as the decision trees found by DT-SLIM*(Weka) and on average the depth of decision trees found by DT-SLIM(Weka) have 34% lower depth. The difference between decision trees found by DT-SLIM(ITI) and DT-SLIM*(ITI) is less definite. While the average difference in depth between decision trees is only 2% in favor of DT-SLIM(ITI), on a per instance level, DT-SLIM(ITI) generates trees with between 2% higher and 20% lower depth than the trees generated by DT-SLIM*(ITI).

Discussion. The trees generated by Weka and ITI vary greatly in depth: Weka-induced trees are, on average, twice as deep as ITI-induced trees. This relationship does not show in the tree size, where Weka-induced trees have, on average, 367 nodes, which is similar to the average size of 320 nodes for ITI-induced trees.

This difference in depth explains the difference in gains

Instance Name	$\|F\|$	$\|E\|$	Weka d	Weka a	DT-SLIM d	DT-SLIM a
australian	1163	552	53.00	0.14	22.00	0.14
ccdefault	211	23955	96.60	0.71	80.00	0.71
haberman	92	240	71.20	0.66	63.80	0.62
hiv schilling	40	2617	18.80	0.81	11.80	0.79
hungarian	330	235	27.00	0.19	9.40	0.59
ida	2195	59998	61.00	1.00	51.00	1.00
objectivity	316	796	36.60	0.55	10.60	0.78

Table 3: Comparison of depth and accuracy on selected instances before and after DT-SLIM(Weka) with d_{800}, c_{800}.

Instance Name	$\|F\|$	$\|E\|$	ITI d	ITI a	DT-SLIM d	DT-SLIM a
australian	1163	552	14.00	0.42	12.60	0.48
ccdefault	211	23955	22.20	0.71	18.40	0.71
hiv 1625	40	1300	13.20	0.85	10.00	0.83
hungarian	330	235	14.00	0.38	9.60	0.59
ida	2195	59998	19.00	1.00	18.00	1.00
kr-vs-kp	37	2556	14	0.98	10.20	0.97
mammog	19	513	13.20	0.68	8.80	0.64

Table 4: Comparison of depth and accuracy on selected instances before and after DT-SLIM(ITI) with d_{800}, c_{800}.

seen in Tables 3 and 4. Here, Weka-induced trees provide more possibilities for depth reduction. In terms of test accuracy, the gains for ITI-induced trees are usually small, while for Weka-induced trees, there are several instances where the test accuracy significantly improved with DT-SLIM. This suggests that without pruning, the Weka-induced trees suffer more from overfitting. Independent of the tree's source, DT-SLIM improved the depth of almost all trees and often by a significant amount.

The results appear robust. We observe only a small influence of the parameter settings. In general the shortest timeout performs worst, while it depends on the instance which of the other two timeout settings perform better. The overall timeout for the whole instance has more impact. For the large instances, DT-SLIM did not reach a plateau within 12 hours, hence a longer timeout enables further improvements.

The new encoding does indeed provide significant additional reduction in the decision tree's depth. As expected, the gains are higher for Weka-induced trees, as the higher depths make DT_pb more applicable. Nonetheless, even for the more shallow ITI trees, the difference in depth is up to 20%, which is a significant gain.

Conclusion

We have presented the new approach DT-SLIM to learning decision trees of small depth, combining standard heuristic methods with exact methods. We facilitated this with (i) a general replacement scheme utilizing new classification categories, (ii) a subtree selection strategy, (iii) a feature reduction heuristic, and (iv) a new partition-based SAT encoding, specifically designed to support non-binary classification and to scale to larger depths.

We have experimentally evaluated this approach on an extensive set of standard benchmark instances, using two different standard heuristics for the initial decision tree, yielding two instantiations of our new approach, DT-SLIM(Weka) and DT-SLIM(ITI). Our experiments show that in almost all cases, a depth reduction is possible, often the reduction is substantial. Our experiments confirm the expectation that, on average, decision trees of lower depth provide higher accuracy. For future work, we propose to extend the DT-SLIM approach to work directly with non-binary features without binarization, or even continuous-valued features by utilizing SMT-encodings.

Acknowledgments

We acknowledge the support from the Austrian Science Fund (FWF), projects P32441 and W1255, and from the WWTF, project ICT19-065.

References

Avellaneda, F. 2020. Efficient Inference of Optimal Decision Trees. In *Proceedings of the The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI-20*. AAAI Press, Palo Alto, California USA.

Bertsimas, D.; and Dunn, J. 2017. Optimal Classification Trees. *Machine Learning* 106(7): 1039–1082. ISSN 0885-6125, 1573-0565. doi:10.1007/s10994-017-5633-9.

Bessiere, C.; Hebrard, E.; and O'Sullivan, B. 2009. Minimising Decision Tree Size as Combinatorial Optimisation. In Gent, I. P., ed., *Principles and Practice of Constraint Programming - CP 2009*, 173–187. Berlin, Heidelberg: Springer Berlin Heidelberg.

Boros, E.; Horiyama, T.; Ibaraki, T.; Makino, K.; and Yagiura, M. 2003. Finding Essential Attributes from Binary Data. *Ann. Math. Artif. Intell.* 39: 223–257. doi:10.1023/A:1024653703689.

Breiman, L.; Friedman, J. H.; Olshen, R. A.; and Stone, C. J. 1984. *Classification and Regression Trees*. Wadsworth. ISBN 0-534-98053-8.

Fichte, J. K.; Lodha, N.; and Szeider, S. 2017. SAT-Based Local Improvement for Finding Tree Decompositions of Small Width. In Gaspers, S.; and Walsh, T., eds., *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, 401–411. Springer Verlag. doi:10.1007/978-3-319-66263-3_25.

Ganian, R.; Lodha, N.; Ordyniak, S.; and Szeider, S. 2019. SAT-Encodings for Treecut Width and Treedepth. In Kobourov, S. G.; and Meyerhenke, H., eds., *Proceedings of ALENEX 2019, the 21st Workshop on Algorithm Engineering and Experiments*, 117–129. SIAM. doi:10.1137/1.9781611975499.10.

Gilpin, L. H.; Bau, D.; Yuan, B. Z.; Bajwa, A.; Specter, M.; and Kagal, L. 2018. Explaining Explanations: An Overview of Interpretability of Machine Learning. In Bonchi, F.; Provost, F. J.; Eliassi-Rad, T.; Wang, W.; Cattuto, C.; and Ghani, R., eds., *5th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2018, Turin, Italy, October 1-3, 2018*, 80–89. IEEE.

Heule, M.; and Szeider, S. 2015. A SAT Approach to Clique-Width. *ACM Trans. Comput. Log.* 16(3): 24. doi:10.1145/2736696.

Hu, X.; Rudin, C.; and Seltzer, M. 2019. Optimal Sparse Decision Trees. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; Alché-Buc, F. d.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Hyafil, L.; and Rivest, R. L. 1976. Constructing Optimal Binary Decision Trees is NP-Complete. *Information Processing Letters* 5(1): 15–17.

Janota, M.; and Morgado, A. 2020. SAT-Based Encodings for Optimal Decision Trees with Explicit Paths. In Pulina, L.; and Seidl, M., eds., *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, 501–518. Springer Verlag. doi:10.1007/978-3-030-51825-7.

Kohavi, R. 1995. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, 1137–1145. Morgan Kaufmann.

Lodha, N.; Ordyniak, S.; and Szeider, S. 2017a. A SAT Approach to Branchwidth. In Sierra, C., ed., *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 4894–4898. ijcai.org. doi:10.24963/ijcai.2017/689. Sister Conference Best Paper Track.

Lodha, N.; Ordyniak, S.; and Szeider, S. 2017b. SAT-Encodings for Special Treewidth and Pathwidth. In Gaspers, S.; and Walsh, T., eds., *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, 429–445. Springer Verlag. doi:10.1007/978-3-319-66263-3_27.

Molnar, C. 2019. *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/. Version from 2021-03-08.

Murthy, S. K. 1998. Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey. *Data Mining and Knowledge Discovery* 2(4): 345–389. doi:10.1023/A:1009744630224.

Narodytska, N.; Ignatiev, A.; Pereira, F.; and Marques-Silva, J. 2018. Learning Optimal Decision Trees with SAT. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 1362–1368. International Joint Conferences on Artificial Intelligence Organization. doi:10.24963/ijcai.2018/189.

Olson, R. S.; La Cava, W.; Orzechowski, P.; Urbanowicz, R. J.; and Moore, J. H. 2017. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining* 10(1): 36. ISSN 1756-0381. doi:10.1186/s13040-017-0154-4.

Pisinger, D.; and Ropke, S. 2010. Large neighborhood search. In *Handbook of metaheuristics*, 399–419. Springer.

Podgorelec, V.; Kokol, P.; Stiglic, B.; and Rozman, I. 2002. Decision Trees: An Overview and Their Use in Medicine. *Journal of Medical Systems* 26(5): 445–463. doi:10.1023/A:1016409317640.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann. ISBN 1-55860-238-0.

Ramaswamy, V. P.; and Szeider, S. 2021. Turbocharging Treewidth-Bounded Bayesian Network Structure Learning. In *Proceeding of AAAI-21, the Thirty-Fifth AAAI Conference on Artificial Intelligence*.

Utgoff, P. E.; Berkman, N. C.; and Clouse, J. A. 1997. Decision Tree Induction Based on Efficient Tree Restructuring. *Machine Learning* 29(1): 5–44.

Verhaeghe, H.; Nijssen, S.; Pesant, G.; Quimper, C.-G.; and Schaus, P. 2020. Learning optimal decision trees using constraint programming. *Constraints* 25(3-4): 226–250.

Verwer, S.; and Zhang, Y. 2019. Learning Optimal Classification Trees Using a Binary Linear Program Formulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 1625–1632.

Zhu, H.; Murali, P.; Phan, D.; Nguyen, L.; and Kalagnanam, J. 2020. A Scalable MIP-based Method for Learning Optimal Multivariate Decision Trees. In *Advances in Neural Information Processing Systems*, volume 33, 1771–1781. Curran Associates, Inc.